Assignment2 - Step 2

Phoebe Zhou

September 2025

Naive RAG Pipeline and Documentation

For this step, I built a naive Retrieval-Augmented Generation (RAG) pipeline using the Mini Wikipedia dataset. The goal was not to make the most sophisticated system yet, but to get a working end-to-end baseline that retrieves relevant passages and generates answers grounded in them.

1. **How the pipeline works.**

   The workflow follows the standard RAG pattern:

   1. Embed the corpus. Every passage in the Wikipedia corpus is converted into a vector using a sentence embedding model. I chose all-MiniLM-L6-v2 from the Sentence-Transformers library because it's lightweight, efficient, and recommended for this assignment.
   2. Index the embeddings. To support fast nearest-neighbor search, I store the vectors in an index. If FAISS is available, the code uses it; otherwise, it falls back to a NumPy-based similarity search. This flexibility avoids environment issues (for example, FAISS can be finicky on macOS ARM).
   3. Retrieve relevant passages. At query time, the question is embedded in the same space and compared against the corpus. The top-k most similar passages are returned, ranked by similarity score. In the naive version, I stick to using the top-1 passage so evaluation reflects raw retrieval quality.
   4. Generate an answer. A text-to-text model (default: flan-t5-base) is prompted with the retrieved passage and the question. The generator then produces an answer that should ideally match the gold reference.

2. **Prompting strategies.**
   Although this is a "naive" pipeline, I included three prompt templates to test different behaviors:

   - Instruction: a direct prompt ("Answer the question using ONLY the context").
   - Persona: positions the model as a helpful tutor, nudging it toward concise, student-friendly responses.
   - Chain-of-thought (CoT): encourages step-by-step reasoning, which may help on multi-hop questions but risks extra verbiage.

Even though the components are simple, having a clean baseline is critical. It tells us how well a straightforward retrieval + generation pipeline performs before we start adding more advanced steps like reranking or query rewriting. For example, if performance is already high with top-1 retrieval, we know later improvements will need to address subtle weaknesses like answer faithfulness rather than brute recall.

3. **How to run.**

From the project root:

```
python -m src.naive_rag --run --top_k 1 --prompt_style instruction
python -m src.evaluation --pred_path results/predictions_naive.jsonl --gold_path data/evaluation/gold.jsonl
```

This simple pipeline gives us a working RAG system. It's not optimized yet, but it lays a solid foundation: we can now measure how much each enhancement (like reranking or query rewriting) actually helps compared to this baseline.